THE Data Modeling Handbook

A BEST-PRACTICE APPROACH TO BUILDING QUALITY DATA MODELS

Michael C. Reingruber and William W. Gregory

The Data Modeling Handbook

A Best-Practice Approach to Building Quality Data Models

Michael Reingruber
William W. Gregory

A Wiley–QED Publication
John Wiley & Sons, Inc.
New York • Chichester • Brisbane • Toronto • Singapore

Publisher: Katherine Schowalter
Editor: Rich O'Hanley
Managing Editor: Maureen B. Drexel
Text Design & Composition: Publishers' Design and Production Services, Inc.

Library of Congress Cataloging-in-Publication Data:

Reingruber, Michael C.
The data modeling handbook: a best-practice approach to building quality data models / Michael C. Reingruber, William W. Gregory.
p. cm.
"A Wiley-QED publication."
Includes Index
ISBN 0-471-05290-6 (cloth)
1. Database design 2. Data structures (Computer science)
I. Gregory, William W.
QA76.9.D26R45 1994
005. 74—dc20 94-12669
CIP

Printed in the United States of America
10 9 8

4

Attribute Rules

4.1 ATTRIBUTE CHECKLIST ITEMS

The Attribute Checklist presents the set of characteristics a data modeling team should gather and record about each attribute to ensure that a complete set of requirements is represented in the data model. Our checklist directly supports the completeness aspect of data model quality.

The list presented here has been made without regard to the limitations or capabilities of any single CASE tool. Rather, it represents those things that we find are essential data requirements, and are necessary to begin database design.

☑ Name

Every attribute shall be given a name that uniquely identifies it. See Attribute Name Rule later in this chapter for further explanation.

☑ Description

Every entity shall have a description that permits it to be correctly interpreted by anyone reviewing the model. The description should be created at the time the attribute is identified and named, and should follow the guidelines established under the Attribute Description Rule found later in this chapter.

☑ Type

An attribute must be categorized as one of two types, either a key attribute or a non-key attribute. This characteristic refers to an attribute's actual use in the model, rather than its potential to serve as a key attribute.

Key attributes serve as primary keys in the entity in which they originate (are first defined), and as either primary or foreign (nonprimary) keys in any entity to which they migrate in support of entity relationships. A primary key is an attribute whose values can be used to uniquely identify occurrences of an entity. Foreign key values are not necessarily unique within the entity. We will discuss primary and foreign keys under the Key Role characteristic a little later. Primary keys are implemented with a unique index to a data structure.

Non-key attributes contain the bulk of the information in a data model. Unlike key attributes, their values do not have to be unique across all occurrences of the entity they have been assigned to through the data modeling process. A non-key attribute can appear in only one entity in the data model. Attributes that are identified as candidate keys (either singularly or in combination with other attributes) but are not selected to serve as primary keys should be classified as non-key attributes. Certain non-key attributes may be selected as access paths during database design, and are often referred to as selection attributes or secondary keys. Despite the latter term, these are generally implemented in data structures using non-unique indices (Finkelstein, 1989).

☑ Edit Rules

Each attribute should be described with one of the edit rules shown in Table 4.1. Edit rules were first documented by Clive Finkelstein, and he provides an excellent description of their use in *Information Engineering: Strategic Systems Development* (1992). An edit rule defines (1) whether a value is required for the attribute upon creation of an instance of the entity in which it appears, and (2) whether the value, once provided, can be updated for that instance of the entity. Edit rules are an essential descriptive characteristic for database and application design. For example, attributes that are 'add now' (see the table below) will need to be implemented as NOT NULL columns in an SQL CRE-

Table 4.1. Edit Rules (from Finkelstein, 1989)

Edit Rule	*Definition*
Add now, modify allowed	A value for the attribute must be provided at the time a new instance of the entity is created. The value of the attribute may be changed one or more times after it is created.
Add now, modify not allowed	A value for the attribute must be provided at the time a new instance of the entity is created. The value of the attribute cannot be changed once it is created.
Add later, modify allowed	A value for the attribute may or may not be inserted at the time a new instance of the entity is created. The value of the attribute may be changed one or more times after it is created.
Add later, modify not allowed	A value for the attribute may or may not be inserted at the time a new instance of the entity is created. The value of the attribute cannot be changed once it is created.

ATE TABLE statement, or the equivalent in other database definition languages.

Primary keys always have a value of "add now, modify not allowed." The edit rule for a primary key migrated to another entity as a non-primary foreign key will depend on the nature of the relationship with the parent entity. An optional relationship to the child entity would translate to "add later," while a mandatory relationship would translate to "add now."

☑ Domain

Every attribute must eventually be assigned a domain of permitted values. Domains need not be determined immediately upon identification of an attribute, but certainly will need to be established before the model can be considered completely stable.

Attribute domains consist of at least one, and possibly two elements:

A **general domain**, also known as a data type, which describes the manner in which the values are represented. Commonly used general domains include alphanumeric, real numbers, integer only,

Boolean, and other standard information processing domains. A general domain must be defined for each attribute. The selected general domain should be the one that represents values of the attribute in a meaningful manner to the user. For example, a person's name could be represented using a hexadecimal coding scheme in place of alphabetic characters, but its utility for users of the data would be severely diminished. The objectives of defining a general domain for each attribute are to select the domain with greatest utility for users, and the one that can be readily translated into data types available in potential or target database management systems. Data models increasingly contain complex data types, such as those required to support sound, digital video, or photographs. See Modeling Complex Data Types in Chapter 9 for further discussion of complex data domains.

General domains may be bounded with high and low range limits. The decision to identify a range of values within a general domain should be done in consultation with the users, and only after careful consideration of both current and future potential values.

A **specific domain**, also known as an enumerated domain, prescribes a specific set of values that are valid and permitted for the attribute. For example, the attribute *property type* in a model for a real estate firm might have the values "townhouse, single family detached, duplex, apartment, . . ." We sometimes refer to these as "static values," since the values in an enumerated domain tend to be relatively stable over time.

☑ Abbreviation(s)

If any abbreviations are used in the attribute name (often required due to limitations of CASE tool in use) or in the attribute description, then it is important to document the abbreviation and a full definition of it as part of the overall description of the attribute. Use of abbreviations should be consistent with any conventions established within the data management community of your organization.

☑ Acronym(s)

We do not recommend using acronyms as a substitute for a full attribute name, although their use may be a practical response to limitations imposed by CASE tools. Acronyms suffer from a couple of common problems—they may not be known or understood by parties outside the

business area being modeled, or even by all parties within the area; and acronyms are often not unique. As with abbreviations, if you use acronyms be sure to document them as part of the description.

☑ Key Use

This rule applies only to primary keys. A primary key that is migrated to another entity will serve in one of two possible roles in the child entity: primary key or foreign (nonprimary) key. The key use depends of course on whether the attribute is required to uniquely determine an instance of the entity into which it is migrated. Note that there are specific rules governing the migration of keys, and these are spelled out in detail in this chapter.

☑ Source

This defines whether the attribute is derived or primitive. A derived attribute is one whose value is generated from one or more other attributes in the model according to a specified formula. A primitive attribute is one whose value is not derived within the scope of the model. A primitive attribute may in fact be derived outside of the scope of the model, but its derivation is not performed within the enterprise. For example, suppose Company A acquires products from Company B. Company A records a *product price* on a purchase request. It is not concerned about how that price is determined by Company B; it simply needs to know how much to pay. If we are building a data model for Company A, the attribute is defined as being primitive. However, suppose Company B actually computes the *product price* according to a tiered pricing structure based on the size of the order. If we are building the data model for Company B, the *product price* is derived (from order size, base product price, and price-quantity discount).

☑ Derivation Formula(s)*

If the attribute source is determined to be derived, then the modeling team, with participation of informed business experts, must establish the formula by which it will be derived. It is extremely important that this is documented, to ensure that eventual users have confidence in the quality of data generated. Derivation formulas can be relatively simple, such as *total sales tax* = *total price* × *sales tax rate*, or they can be

* Applies only to derived attributes.

more complex, with adjustments to the formula based on values or conditions associated with the attribute. The formula should explicitly identify any other attribute(s) required to generate a value for the derived attribute. We will revisit derived attributes in greater detail later in this chapter.

☑ Derivation/Attribute Dependency Links*

Derived attributes are almost always dependent on at least one other attribute in the model. This dependency has been referred to as a *derivation dependency* (McClure, 1993). You should maintain documentation of such links as an aid to understanding the derived attribute, and as input to eventual information system planning, design and development.

☑ Requirements Traceability

This is information pertaining to the cause or purpose of the enterprise's interest in the attribute. The requirement may have one of several sources, depending on the nature of the data modeling activity (e.g., reverse engineering, strategic information systems planning, etc.) Traceability may be attained through (1) reference to the source of the need (e.g., citation of a statement, clause, paragraph, or section of a regulation, policy or plan; or, reference to a physical data structure element, program variable, or human interface component in the case of current systems analysis sources); and, (2) mapping to another metadata object created and managed as part of the system lifecycle, such as a planning statement (e.g., critical success factor, goal, objective) or a physical system or system design element (e.g., file, data element, table, program element). In other words, you need to document where the entity came from, and where it is going to be used.

☑ Ownership Authority

You should maintain a record of individuals, functions, or organizations that have authority over each attribute for the following purposes:

Stewards have authority for defining and approving metadata. Individuals participating in the development of a data model are defining, and at least preliminarily, approving metadata. Attribute names,

* Applies only to derived attributes.

descriptions, enumerated values and other characteristics fall under the authority of stewards. Your data administration organization should have policies and procedures for metadata control.

Custodians create and use data. These are the individuals or organizations that have been granted authority within the enterprise, usually through policies and procedures, to perform business operations or processes that result in creation of data. Although access authority is granted through many different means, it will be important to understand *and plan* who within the enterprise has authority to create, read, update, and delete data. The planning of data manipulation will become increasingly important as more and more organizations move toward the distributed, shared data environments inherent to client/server architectures.

4.2 DERIVED ATTRIBUTES

Derived and calculated attributes must be represented in the data model, along with their dependencies to the base attributes used in the derivation or calculation.

Discussion

We begin the discussion of this rule with definitions of some key terms. Like many of the terms used in the information resource management discipline, these have become overloaded through years of use. We are not suggesting that our definitions become the standard; we provide them simply to clarify discussion of the rule.

A *derived attribute* is one that is created by accumulating values of multiple instances of one or more other data attributes and, optionally, performing some additional computation on those values in order to create a value for the subject derived attribute (Inmon, 1989). Thus, it represents an aggregation or summarization of existing information, and introduces a derivation dependency within the model (between the derived attribute and the other attribute(s) from which it is derived). A derivation dependency exists between a derived attribute and the base attribute(s) from which it is derived (McClure, 1993). Figure 4.1 presents an example of a derived attribute and its dependencies.

A *calculated attribute* describes some characteristic about a single instance of the entity, and is generally calculated from another single instance of a related attribute. In the simplified example in Figure 4.2, the attribute *task duration* is a calculated attribute, created by subtracting the completion date from the start date. As with derived attributes,

5

Relationship Rules

5.1 RELATIONSHIP CHECKLIST

The Relationship Checklist presents the set of characteristics a data modeling team should gather and record about relationships to ensure that a complete set of requirements is represented in the data model. Our checklist directly supports the completeness aspect of data model quality.

As with the entity and attribute checklists, the list presented here has been made without regard to the limitations or capabilities of any single CASE tool.

☑ Name*

This is the name of the relationship that meets naming conventions or standards. Some modeling techniques ordain the use of two relationship names, supposedly enabling the reader to understand the interaction of two entities much better than if only one name is used. Relationship names are actually verbs, not nouns, and therefore imply that one of the entities takes action or maintains a certain state relative to the other (and vice versa, of course).

Do not waste your time or risk the wrath of your business expert participants laboring through the creation of relationship names. Rela-

* Optional

tionship names generally add very little, if any, clarity to the model (even when done "properly"), tend to be repetitious, are tedious to define and maintain, and do not aid database design in any way. There is nothing more painful than watching a data modeler facilitate a group of users through a relationship naming exercise. And most analysts and business experts, judging by the models we have seen, do not do a very good job of relationship naming. The old stand-bys "has" and "owns" and "has an interest in" show up everywhere. Once you reach several hundred entities in your model, the prospect of naming relationships becomes rather dreadful. Our recommendation is that you spend your valuable time on some of the truly important aspects of the modeling process. CASE tool vendors have gotten the message, and most either no longer require relationship names or allow the modeler the option to not depict them on the data model diagrams. Unfortunately, several widely used CASE products still impose a requirement that you provide a name before adding the relationship to the model, even though you choose not to display them.

☑ Description*

The description presents the analyst an opportunity to record additional details about a relationship. This information does not duplicate other elements of the relationship; it is simply intended to add additional documentary detail to the model. In most cases, information provided in the relationship description will be available through other objects in the meta-model.

☑ Key(s)

Every relationship must be supported by the migration of a primary key (whether single attribute or multiple attributes). This is the basic precept of referential integrity, and is one of the foundations of relational data modeling. It is extremely important that the relationship be supported by the migration of the entire primary key to the child entity, and that the entire migrated key serve as either a component of the primary key or as a foreign key reference. Migrating composite keys should never be split between primary and nonprimary foreign key use in the child entity. We have outlined rules to this effect in our discussion of primary keys.

---

* Optional

☑ Type [Required by some methodologies]

Certain methodologies categorize relationships according to the role that the migrating keys will play in the child entity. For example, the IDEF1X modeling notation designates relationships as either identifying, nonidentifying, or category. In an identifying relationship, the primary key(s) of the parent is migrated to serve as part of a compound primary key in the child. A nonidentifying relationship results in the migration of the parent entity's primary key as a nonprimary foreign key in the child. Finally, category relationships exist between supertypes and subtypes, with the primary key of the supertype 'migrating' intact to each subtype.

This property actually provides redundant information, since we know the role that the migrated keys are playing (primary or nonprimary) in the "downstream" or child entity. We have found the designation of relationship type to add very little value. In the case of IDEF1X, which we discussed above, we suspect that it emerged as part of that technique's phased approach to modeling, where key structures are not defined until the second phase. If you use a CASE tool such as Texas Instruments' IEF™ or LogicWorks' ERwin™, you may be required to designate a relationship type. Consult your CASE tool requirements.

☑ Cardinality/Optionality

This is also referred to as Degree/Nature or simply Cardinality. The degrees and natures of a relationship are the soul of business modeling. They allow for the precise depiction of business rules. By examining alternative degrees and natures at each end of a relationship, the modeling team can examine alternative business strategies and policies using the model as a visualization technique. We strongly urge you to read Chapter 4 of *An Introduction to Information Engineering* by Clive Finkelstein (1989) for a description of the role relationship degrees and natures can play in examining business rules and strategies.

The rules surrounding allowable combinations and depiction of relationship degrees and natures will vary, in some cases substantially, depending on the modeling notation you have selected. We suggest that you explore several methods before deciding which you are most comfortable with. Your data management policies may dictate use of a specific notation, but continue to look around. While we have chosen to use the most popular notation for this book, there are some interesting alternatives out there. Of particular interest are Clive Finkelstein's use of "optional-tending-toward-mandatory" and the IDEF1X standard's use of degrees with specific values (i.e., instead of "mandatory many,"

you might specify "mandatory 3 to 5," depending on your business rules) (Finkelstein, 1989; FIPS, 1992).

☑ Deletion Integrity Rules

There are three fundamental types of relational integrity—add, update, and delete. Add and update integrity rules are explicitly defined through the relationship cardinality. However, delete rules must be defined separately. There are three alternatives for establishing delete integrity—cascade, disassociate, or disallow. You will find our rule pertaining to delete integrity in Chapter 9.

☑ Relationship Quantity Statistics

Prior to moving to data structure design, you will want to gather some statistics to understand the quantitative aspects of relationships. While other statistics may be gathered, the most critical information for database design, specifically for transaction access path analysis, is the entity ratio for each relationship (min, max, and average). Figure 5.1 illustrates this concept. Current systems analysis should assist you in determining ratios, but participation of business experts will be necessary to confirm the numbers, and to assess any new requirements not found in current systems (whether automated or nonautomated).

☑ Discriminator

An attribute should be established as the discriminator for each supertype-subtype relationship. The discriminator attribute is used to record which

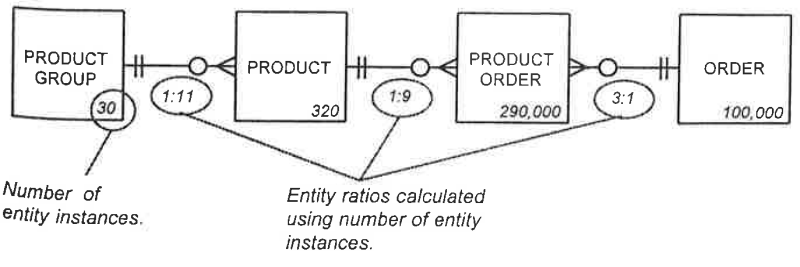


Figure 5.1 Average entity ratios for a set of relationships.

subtype(s) are applicable for a particular instance of the supertype. See Chapter 6 for further discussion of discriminators.

5.2 BALANCED ONE-T0-ONE RELATIONSHIPS

Eliminate balanced one-to-one relationships from the model.

Discussion

A balanced one-to-one relationship has the same cardinality/ optionality combination at both ends of the relationship.

The introduction of balanced one-to-one relationships introduces some problems to a data model.

- A mandatory one-to-one relationship indicates that the two entities are so closely related that one cannot exist without the presence of the other. This type of close tie generally indicates that the two things represented by the separate entities are actually characteristics of the same thing. Let's assume for a moment that we have two entities as depicted in Figure 5.3.

  Recall the definition of functional dependence from Date (1986):

  Given a relation R, attribute Y of R is *functionally dependent* on attribute X of R if and only if each X-value in R has

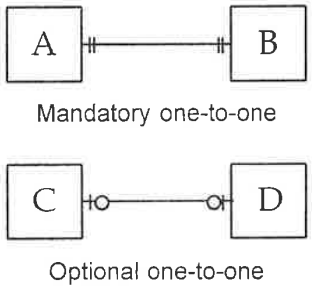


Figure 5.2 Balanced one-to-one relationships.

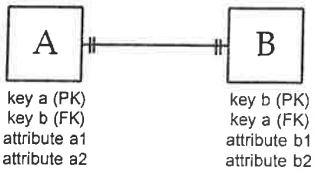


Figure 5.3 Mandatory one-to-one.

associated with it precisely one Y-value in R (at any one time). Attributes X and Y may be composite.

Now back to our model. Attribute *key a* is functionally dependent on *key b* (in entity B, each attribute *key b* value has associated with it precisely one *key a* value at any one time). Likewise, attribute *key b* is functionally dependent on *key a* (in entity A, each attribute *key a* value has associated with it precisely one *key b* value at any one time). That's interesting. We have introduced a case where the nonkey attributes of both entity A and entity B are functionally dependent on both the primary key of A and the primary key of B. For each *key a* value in A, *attribute b1* and *attribute b2* in entity B have associated with it precisely one value. Likewise, for each *key b* value in entity B, *attribute a1* and *attribute a2* in entity A have associated with it precisely one value. The decomposition of A and B into two entities is therefore arbitrary. All four attributes are functionally dependent on *key a* or *key b* (both are candidate keys), and should be placed in a single entity. Selection of a candidate key as primary key for the new entity should be worked out with the business experts. The same logic holds for balanced optional one-to-one relationships. In the case of an optional relationship, the resolution may include the creation of subtype entities to manage optional attributes in the new entity (created through resolution of the balanced relationship).

We have demonstrated this situation assuming that the entities had different primary key attributes, the migration of keys was performed in order to maintain referential integrity, and the keys were migrated as nonprimary foreign keys. Suppose now that they

were migrated as primary keys. Well, we would have two entities with the same primary key attributes, which violates one of our fundamental rules about primary keys (see Chapter 4 for further discussion of primary key rules).

- Balanced one-to-one relationships prevent the modeling team from establishing the data dependency path.
- Balanced one-to-one relationships introduce unnecessary redundancy into the model, and therefore raise the risk of introducing inconsistency.
- They may mask the actual logic that the modeling team was striving for, or lead to confusion over relationships and business rules.

The definition of the separate entities may come about during the course of modeling for some very understandable reasons:

- They were created by separate modeling teams, followed by an attempt to integrate the results.
- In the case of the optional one-to-one relationships, it is possible the modeling team was attempting to show that certain attributes are optional at the time of creation of an instance of the entity.
- By using the optional one-to-one relationship, the team may have been trying to indicate that the entities are mutually exclusive subsets of the same logical concept (a situation best handled with a subtype structure).
- The entities were identified in conjunction with functional or process analysis, and have been artificially separated because they are used to perform different functions or are involved in different processes.
- The attributes in the separate entities are owned by or controlled by different organizations.

These are just a few of the motivations that a modeling team may use to argue the case for employing one-to-one relationships. However, in all cases the separation of attributes is arbitrary, and therefore not repeatable consistently across the model. Remember that in order to maintain a model that can be used across an enterprise, the model must be built using techniques that lead to repeatable results. With balanced one-to-one relationships, there is no basis for dividing the attributes between the entities in a consistent fashion.

If you run across balanced one-to-one relationships, attempt to merge the participating entities into a single entity, or into a supertype/subtype

structure in the case of the optional one-to-optional one relationship. The examples provided in the next section illustrate resolution scenarios.

Example

For our example, assume we have a small business that is modeling the management of its sales force. The company has defined a series of territories, and *currently* has a sales representative assigned to each. The company wants to track its sales by territory, since it has spent a great deal of time studying regional demographics and wishes to track buying habits accordingly. The sales manager, however, needs to track individual performance of each of her sales representatives. The model fragment shown in Figure 5.4 emerged from the modeling sessions (we have shown key attributes only).

It is clear that the modeling team has captured the *current* business climate from an organizational and functional perspective. In addition to the technical problems that the one-to-one relationship introduces, it also represents an artificially constrained business environment as well. The near-term solution to the modeling problem posed by the relationship might be to create a single SALESPERSON entity. However, the team should also look forward to the future sales strategies and policies of the company. How will sales territory be defined in the future? Will territories always be defined in terms of salespeople, or will they take on demographic or geographic connotations? Is it possible that a SALESPERSON could cover more than one territory? Could we have two salespersons covering a sales territory? Do we want to track different salespersons' performance in the same territory as they are reassigned? These are conceptual questions that the presence of a one-to-one relationship should prompt.

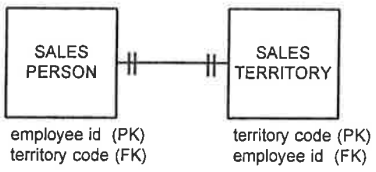


Figure 5.4 Invalid one-to-one relationship.

5.3 MANDATORY-ONE-TO-OPTIONAL-ONE RELATIONSHIPS

Carefully examine mandatory-one-to-optional-one relationships in the model.

Discussion

Technically, a mandatory-one-to-optional-one relationship is beset with the same problems that plague relationships discussed in the Balanced One-to-One Relationships rule.

The difference between the cardinality set shown here and those shown in our other rule is that we can define the data dependency chain between entities F and E as pictured above. An instance of entity F must clearly be created at the same time or earlier than an instance of entity E. In other words, we can have F without E, but not E without F. As a result, we have found that this cardinality set is a warning flag for conceptual quality problems, and have chosen to address it separately.

Guidelines for Resolving Optional-One-to-Mandatory-One Relationships

- Consider whether entity E represents a subtype of F. Does entity E actually represent one of many categories of F, but is the only one with additional attribute requirements? Many methodologies discourage the depiction of subtype entities that have no attributes other than the inherited primary key, since these may be more effectively handled by defining, and *documenting as part of the model*, a domain of values for the discriminator attribute. There is a danger in using the mandatory-one-to-optional-many cardinality to accommodate such a situation, since the domain of other subtypes may be lost if not correctly documented. Is there an attribute in F whose value triggers a requirement for populating E? If so, this

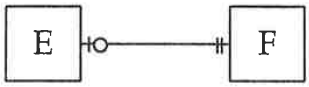


Figure 5.5 Unbalanced one-to-one relationship.

attribute serves as a discriminator, and indicates that a generalization hierarchy structure is appropriate. See Chapter 6 for further discussion of generalization hierarchies.

- Determine if E has been modeled to accommodate a time delay between the creation of F, with its primary key and non-key attribute set, and the creation of instances of the attributes in E. This situation does *not* require the identification of a separate entity to handle what might be referred to as "add later" attributes.
- Determine if E has been modeled to accommodate an expected difficulty in actually obtaining the information represented by the attributes of E. If the users expect to gather values for only a small percentage of some of F's attributes, they may have been tempted to move them to E. The logical model should not reflect such considerations, although recording such expectations will certainly aid the database design team.
- Has the cardinality for one or both of the entities been improperly defined? Check the key structure to see if only one of the two entities has a migrated attribute; that might indicate that the cardinality was incorrectly recorded.
- Review any applicable business rules to ensure that the model is conceptually correct.

Examples

The example we have selected (Figure 5.6) comes from a human resources department model, and is perhaps one of the most extreme examples of a timing difference influencing model development. The objective of the model was to separate current and former employees, since these are obviously subject to different sets of rules and different information needs. Unfortunately, the model fragment that resulted is incorrect. The modeling team has separated FORMER EMPLOYEE from EMPLOYEE based on the assumption that employee termination date is optional. However, it is not optional. Every employee will eventually be terminated for one reason or another (retirement, death, resignation, involuntary termination). The issue is one of timing. Therefore, the attribute *employee termination date* is not optional in EMPLOYEE—it will eventually apply to all employees. The modeling team has been overly influenced by the often substantial difference in the time an instance of EMPLOYEE is created and the time a value for *employee termination date* will be added. That timing difference does *not* make *employee termination date* an optional attribute. Remember that we are building a logical data model. Decisions about how best to implement this will have to be made, just not now.

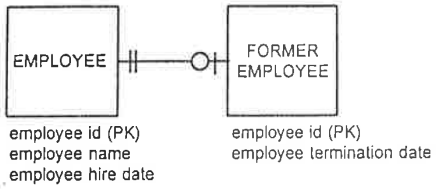


Figure 5.6 Unresolved relationship.

Our resolution is depicted in Figure 5.7. Note that *employee termination date* is now originating in EMPLOYEE.

A second resolution is depicted in Figure 5.8. This resolution increases the flexibility of the model. We have added an *employee status* discriminator to identify employees who are currently employed, have retired, have been fired, or have some other status with the company. Note that *employee hire date* and *employee termination date* become "hire" and "terminated" status values in *employee status*, combined with corresponding *employee status effective date* values. There are obviously many alternative solutions, and the correct one for your organization lies in your business policies and the needs of your business experts.

Our second example involves information that the users expect will be difficult to obtain. The users maintain data on airports around the country for a regional airline. They gather information from a wide variety of sources, but have had considerable difficulty in the past obtaining data on runway surfaces at small airports. The airline's pilots need access to this information in occasional emergencies, when they must land at unfamiliar airports. As a result, they have modeled RUNWAY SURFACE information separate from RUNWAY information.

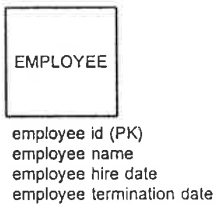


Figure 5.7 Resolved optional relationship problem.

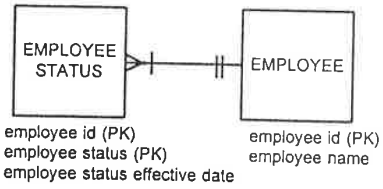


Figure 5.8 A second resolution strategy.

Unfortunately, this suggests that some runways do not have surfaces. It would be logical to assume that *every* runway has a surface of some kind. Once this kind of arbitrary division is introduced, it tends to cause additional problems. Notice *runway surface condition* has crept out to RUNWAY SURFACE, although this information is not necessarily dependent on knowing the exact type of surface. The model has been corrected in Figure 5.10.

5.4 OPTIONAL-ONE-TO-MANY RELATIONSHIPS

Resolve relationships where the parent entity is optionally related to the child entity as depicted in Figure 5.11.

Discussion

The existence of relationships of the type shown in Figure 5.11 are generally indicative of incomplete modeling. They present some interesting technical problems, which we will discuss momentarily, and they tend to

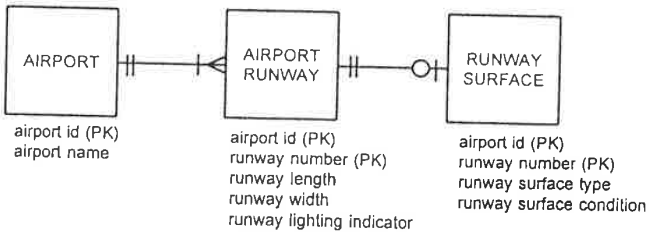


Figure 5.9 Unresolved relationship.

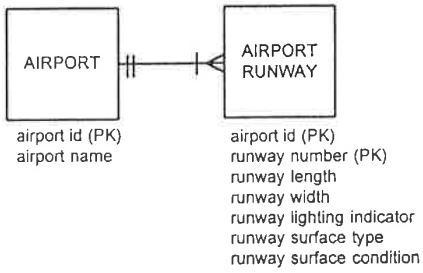


Figure 5.10 One possible resolution to Figure 5.9.

obscure important business logic. Resolve these by whatever means is appropriate to ensure that the relationship from the child entity (in this case both B and D) to the parent entity (A and C) is mandatory. This will improve the depiction of business logic, increase the stability of the model, and aid in the creation of sound data structures.

One might argue that there is nothing wrong with the relationships you see in Figure 5.11. Were we to add keys, we would migrate the primary keys of entities A and C (the parents) to entities B and D (the children), respectively, as *nonprimary foreign keys*. There would be a problem if we attempted to migrate those keys as primary keys in B and D, however. Recall that one rule of primary keys is that they are never optional. Since our relationships indicate that B and D are optionally

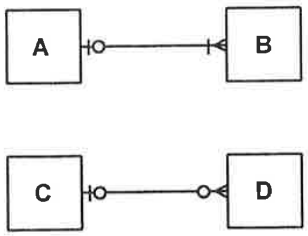


Figure 5.11 Optional-one-to-many.

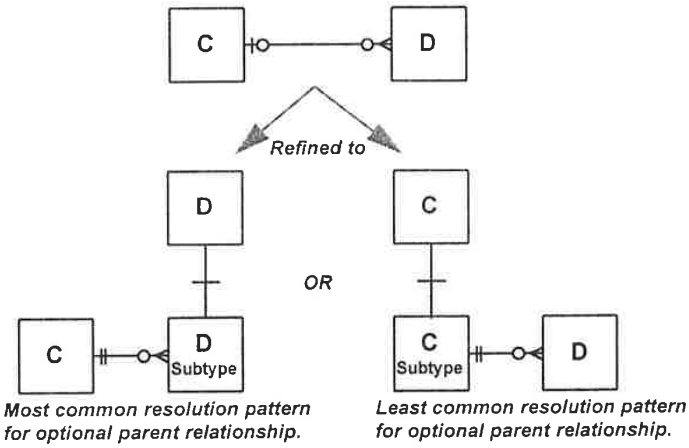


Figure 5.12 Optional parent resolution strategies.

related to A and C, respectively, migration as primary keys is out of the question. The keys for A and C could be migrated as foreign keys. However, remember that our optional attribute rule stipulates that we want to ensure that any attribute, including foreign keys, placed in an entity should be mandatory for all instances of that entity. Ignoring the possibility that the relationship was misrepresented and was meant to be a many-to-many, there are two resolution strategies that are used to resolve this type of relationship (Figure 5.12).

The selected resolution strategy will depend, of course, on the business rules applicable to the individual situation. Note that the resolution strategy on the left is by far the most common, but both are possible results. In either case, we recognize a distinguishing characteristic about C or D that reflects the business rule for a mandatory relationship.

Example

For our example, we will revisit an example we used earlier to describe optional attributes. MATERIAL TYPE represents a standard stock item held in a company warehouse. Some of the materials held are considered

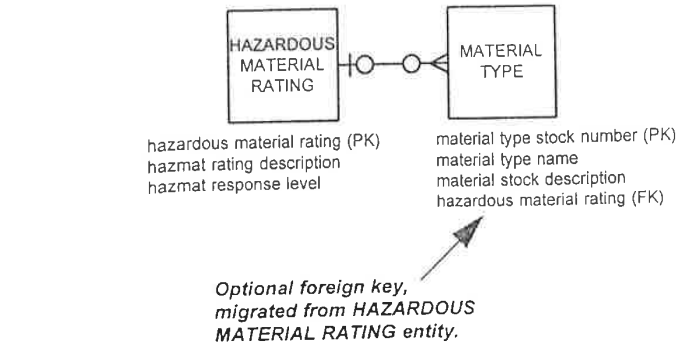


Figure 5.13 Unresolved optional relationship to parent entity.

hazardous (flammable, corrosive, poisonous), and the company has established a hazardous material rating and response procedure to ensure proper handling and rapid response to incidents involving the materials.

Only certain types of material are considered hazardous, with the result that an optional relationship nature exists to the parent entity HAZARDOUS MATERIAL RATING. The business rules and information requirements pertaining to management of hazardous waste only apply to those types of material deemed hazardous. In order to refine the model to precisely depict the business rules, and to position the design team to develop a better database design, the HAZARDOUS MATERIAL entity has been created in Figure 5.14. This resolution strategy follows the general pattern discussed above.

5.5 MANY-TO-MANY (NONSPECIFIC) RELATIONSHIPS

Resolve many-to-many relationships early in the modeling cycle.

Discussion

A many-to-many association, also known as a nonspecific relationship, cannot be implemented in any practical manner in a data structure. While this should be reason enough to resolve it, there is an added incentive to do so early in the data modeling effort. In fact, we recom-

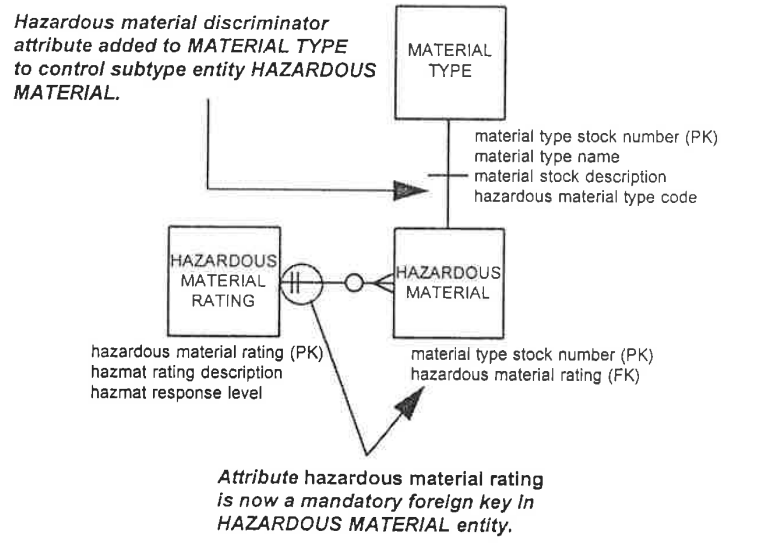


Figure 5.14 Resolved optional relationship to parent entity.

mend that you resolve a many-to-many relationship into an associative entity as soon as the relationship degree and nature have been determined. Why the rush? For several reasons:

- The associative entity may imply additional rules or business logic that would have been overlooked if the many-to-many had not been resolved. In particular, we have found that subtype hierarchies tend to emerge from associative entities as business rules are precisely defined, and that these are overlooked in models where many-to-many relationships are not resolved early in the analysis. If you want to build models that fully capture the business rules of the enterprise, then resolve nonspecific relationships early. Also, we have found that a large portion of management control data requirements, usually in the form of derived data attributes, occur in associative entities. Refer to our discussion on derived attributes in Chapter 4. These might be overlooked, or at least underanalyzed, if nonspecific relationships are not resolved.

- Many-to-many relationships make the determination of data dependencies difficult, limiting your ability to identify logically complete subsets of the model. Recall our discussion on data dependency analysis in Chapter 2, and how it serves as an efficient, reliable, and repeatable method for dividing large data models into cohesive subsets. If you have many-to-many relationships in your model, you will have difficulty determining data dependency paths (associative entities form a large proportion of the end-points necessary to begin data dependency analysis).
- Your model will remain unstable until the many-to-many relationship is resolved. It will not be normalized, and your documentation of the model will not be complete.

The sole exception to the early resolution strategy is if you are working with executives or senior staff members, and you neither want or need to include detail in your model. However, be aware of the fact that if you are modeling management level information, many of the derived data attributes of interest to that group reside in associative entities.

Resolve many-to-many relationships by creating an associative entity that maintains the degrees and natures of the original relationship as shown in Figure 5.15.

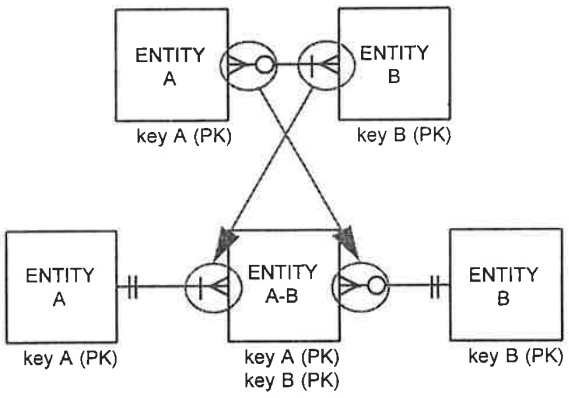


Figure 5.15 Strategy for resolving many-to-many relationships.

The associative entity formed as a result of this action (ENTITY A-B) will have as its primary key the combined migrated primary keys of the original two entities (ENTITY A and ENTITY B). Note that the relationship degrees and natures from the associative to the other two entities are *always* mandatory-one. Like any other modeling construct, there are occasions when the key structure for an associative entity can become more complex. However, this strategy will serve as a good starting point for further modeling around the associative entity.

Example

Assume we are modeling a university registrar's function. Our model has been developed to the point shown in Figure 5.16. The business experts express some concern when they view the model. "How will I be able to tell which student has enrolled in which course, and what the roster of students is for each course? The model doesn't allow me to do that, does it?" And now a new data requirement emerges as well. The business experts in the registrar's office are concerned about transcripts (grades) as well as enrollment. "Where does the attribute for a student's grade in the course belong in our model?" Well, the model fragment in Figure 5.16 simply doesn't do the job any more.

We resolve the many-to-many relationship using an associative entity (see Figure 5.17).

Now the modeling team sees the relationship more clearly, and guess what? That's right, they begin to form additional questions about the business rules, ones that were not apparent before. For example,

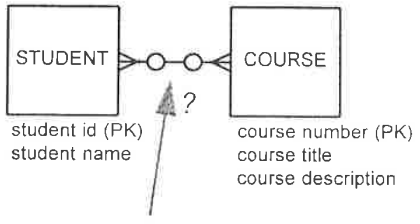


Figure 5.16 Unresolved many-to-many relationship.

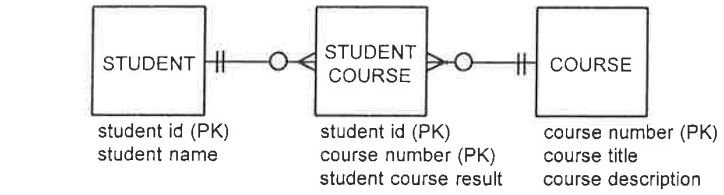


Figure 5.17 Resolved many-to-many relationship.

one member of the team raises the issue about students who take a course more than once, in order to improve their grade in a required prerequisite. The current key structure does not support this requirement. And what about students who enroll but withdraw without a grade penalty, but who are still required to pay part or all of the tuition? And does the registrar need to be able to determine a student's status in relation to the course (i.e., enrolled, completed, withdrew, etc.)? Are dates important to the student course relationship? As you can see, the modeling team can now focus on the student-course relationship, and extend the model by continuing to ask about business rules. We probably would have overlooked some or maybe all of the questions if we had not resolved the many-to-many relationship. Most modeling teams focus on entities and attributes, not relationships, when trying to define new data requirements. Therefore, resolve nonspecific relationships early in your modeling cycle, and use the associative entity as a springboard for further analysis.

5.6 CIRCULAR REFERENCES

Circular references are not permitted.

Description

A circular reference introduces a cyclical relationship into the data model, and should be avoided at all costs. Circular relationships cause problems in the migration of keys, and they prevent the modeler from establishing proper data dependency sequences (Finkelstein, 1989).

Existence of a cyclical relationship indicates that one or more of the following conditions has occurred.

- The modeling team has not clearly or correctly captured the business logic. The team should reevaluate this portion of the model, examining related business rules and policies to determine the correct requirements.
- The cardinality of one or more of the relationships has been accidentally reversed either during the modeling session or during entry to the CASE tool.
- The cardinality for one end of a relationship has been incorrectly recorded. For instance, a many-to-many relationship was recorded as a one-to-many.

Example

Our example comes from a strategic data model created for XYZ Manufacturing Corporation, and addresses the first causal condition described above. Note that we have elected to show only key attributes in this example. XYZ (an ORGANIZATION) offers each of its products and services (OFFERING) in many MARKETs around the world. In each market, it competes against many other ORGANIZATIONs, each of which has many OFFERINGs of its own. The model seems to support the logic of these statements, but there is clearly something amiss. The circular nature of our logic leads to additional questions.

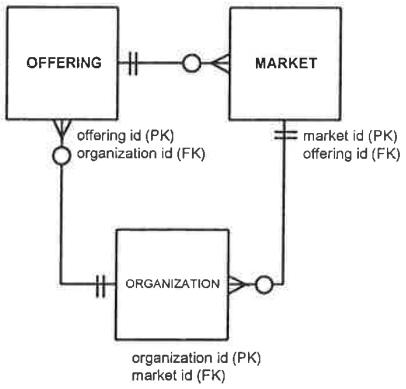


Figure 5.18 Unresolved circular reference.

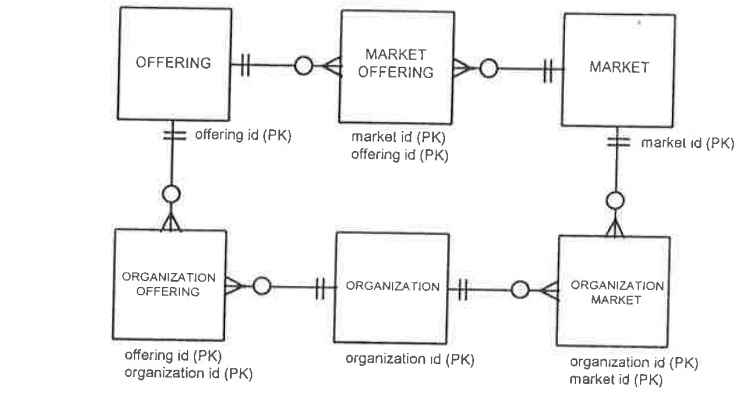


Figure 5.19 Resolution of circular reference.

Now let's examine some refinements to the model. First, we'll reverse our logical progression through the model. Each ORGANIZATION operates in many MARKETs (by virtue of having many OFFERINGs in many MARKETs, as we discussed earlier). And each MARKET has many OFFERINGS (e.g., competing products and services). Finally, depending on how we have defined OFFERING, the same OFFERING may be marketed by many ORGANIZATIONs. The result is a refined model that completes the logic that was captured in the first attempt.

5.7 TRIADS

Eliminate triads from the data model.

Description

A triad is a redundant relationship consisting of two dependency paths between the same two entities (FIPS, 1992). One of the paths consists of a direct relationship between the entities, while the other involves relationships to entities other than our two subject entities (see Figure 5.20). Triads emerge during the normal course of modeling, as teams attempt to precisely represent business policies that overlap or are contradictory.

Triads should be eliminated from the data model because they introduce redundant information, which may eventually lead to problems with database consistency.

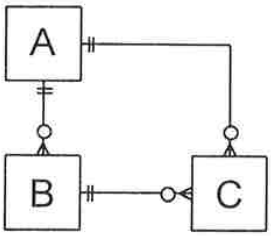


Figure 5.20 General structure of a triad.

If a triad has been created by the team, then simply resolving it in the data model may not be adequate. The existence of a triad may indicate that contradictory policies exist. Before correcting the model, examine the business statements that were used as a basis for that portion of the model, and review session notes to determine the rationale for constructing both dependency paths.

Example

In our example, we have a model fragment that records data requirements and rules pertaining to the monitoring of security alarms. The security firm has established a set of alarm classes (perhaps according to the nature of the installation under surveillance), and has a policy regarding response time for each class. A set of alarm codes exist within each class that provide instructions on how to respond. Finally, individual alarms are monitored, and the firm's performance against each alarm is measured. The firm has established a performance objective relating to beating the response time maximum. The latter objective may have caused the modeling team to create a relationship between ALARM CLASS and ALARM, to ensure that each alarm can be traced to the alarm class to which it belongs. However, the classification policy led to the construction of the ALARM CLASS-ALARM CODE-ALARM dependency path. The model fragment is a triad construct.

According to one of Armstrong's axioms (Date, 1986) on functional dependencies:

If A → B and B → C, then A → C.

You may recognize this as the "transitivity rule." In the case of our example, we have the following dependencies:

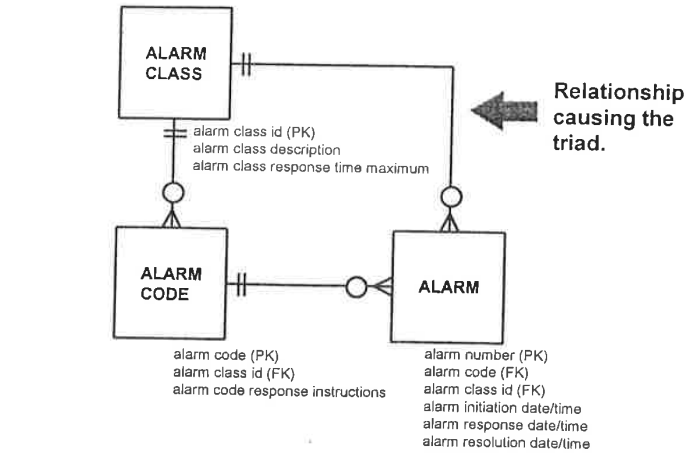


Figure 5.21 Model fragment containing a triad.

alarm number → alarm code
alarm code → alarm class

Therefore, the dependency alarm number → alarm class is "logically implied" (Korth and Silberschartz, 1986), and need not be designated with a separate relationship.

Based on this analysis, the resolution of our example triad is relatively straightforward. Note that there is a transitive dependency between ALARM CLASS and ALARM through ALARM CODE. Therefore,

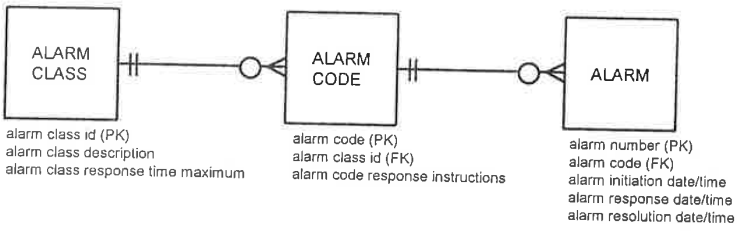


Figure 5.22 Model fragment corrected to eliminate triad.

ALARM can be readily traced to its ALARM CLASS according to the ALARM CODE to which it is assigned. If ALARM CLASS to ALARM CODE were many-to-many, we would have a very different situation. You can confirm this by examining the key structures in the model. The corrected model fragment is shown below. Note the change in the key structure for ALARM.

5.8 MORE THAN ONE RELATIONSHIP BETWEEN TWO ENTITIES

No two entities should have more than one relationship between them.

Discussion

We have encountered several experienced modelers who insist that multiple relationships add important and necessary information to the model. Fortunately, most data modelers have already arrived at the conclusion that multiple relationships between the same two entities actually introduce confusion into a model, rather than adding clarity. In fact, many, if not most, CASE tools prevent the user from making this kind of mistake.

In case there are still some diehards out there, we present some arguments in favor of this rule.

- Use of multiple relationships introduces the potential for conflicting cardinalities. A requirement for differing cardinalities suggests that there are logical subsets of the two entities (subtypes) that are governed by distinct sets of rules. The model needs to be refined and the hidden logic flushed out. Our first example addresses this condition.
- Multiple relationships tend to represent process logic, rather than data requirements and actual relationship rules. Our second example illustrates this case.
- Use of multiple relationships can also obscure or confuse other aspects of the data logic, since their use gives the appearance of comprehensive analysis. The opposite is true, as we can see by both examples.
- Modeling teams may attempt to document the different *reasons* that two entities may be related. Use of multiple relationships to accomplish this is a case of using the wrong tool for the job. We suggest documenting such information in the Relationship Description, or through use of a Reason Entity if an associative entity is involved. Our third example illustrates this type of situation.

Examples

Conflicting Cardinalities In this example we see how easy it is for conflicting cardinalities to invade a data model if multiple relationships are permitted. In our hypothetical organization, two separate data modeling exercises have been underway. The first involves the division level staff, who created a strategic model for their enterprise. This effort was followed up by a priority project in the Personnel Management Subject Area (identified and scoped during the strategic project). The division staff has clear strategies in mind with regard to personnel, productivity, and workforce management. These are driven by their business objectives. The modeling team for the Personnel Management area has a different set of strategies, as we can see in the model. Their reality differs from the perspective of the division staff, resulting in conflicting business rules.

If we let this conflict continue without resolution, the organization ends up with two problems. First, the requirements for information systems development will be unclear. Even if these entities are used in separate systems, an EIS for the division staff and a personnel management system, there is every likelihood that they will draw data from shared databases. But we will have introduced contradictory rules into the enterprise data architecture, and at some point will encounter problems summarizing or sharing information. Second, and perhaps more important, the model indicates a potentially serious gap in communication between division management and one of their key operational areas. Remember, the model is a representation of the actual enterprise. Our example model indicates contradictory strategies and policies exist. The business is going in separate directions.

While our example may be somewhat trivial, the reader should be able to gain a sense of the serious problems that can arise for an organization due to the cumulative effects of many contradictory rules or policies. The modeling team has a tremendous opportunity to help bring the enterprise into alignment by bringing the two groups together to discuss the different views, and to come to a mutual understanding on the business rules and strategies.

Our resolved model, shown in Figure 5.24, reflects the kinds of changes that modeling teams often encounter when attempting to resolve cardinality conflicts for multiple relationships (either before they get into the model, or after). In this case, the model manager brought the two groups together, with the result that the division staff reiterated its desire to have at least one person assigned to each job at all times. Since the personnel department could not ensure that an employee would be available to step into the job, a new strategy was devel-

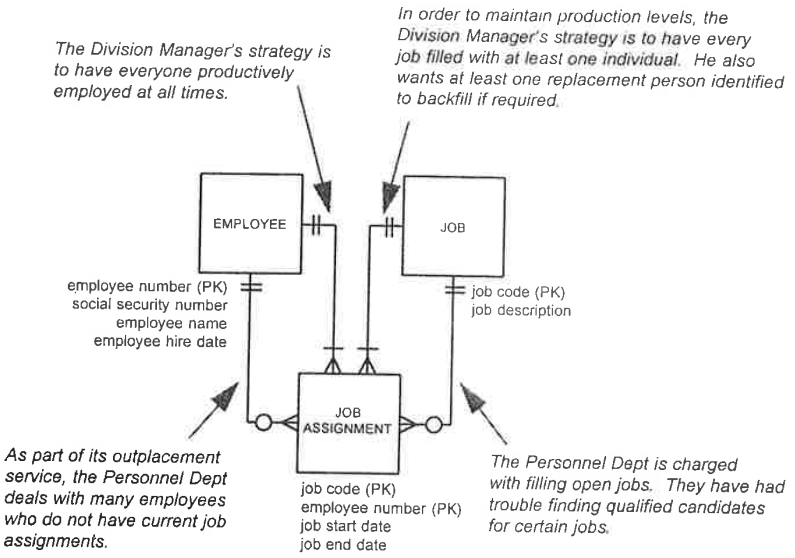


Figure 5.23 Multiple conflicting relationships.

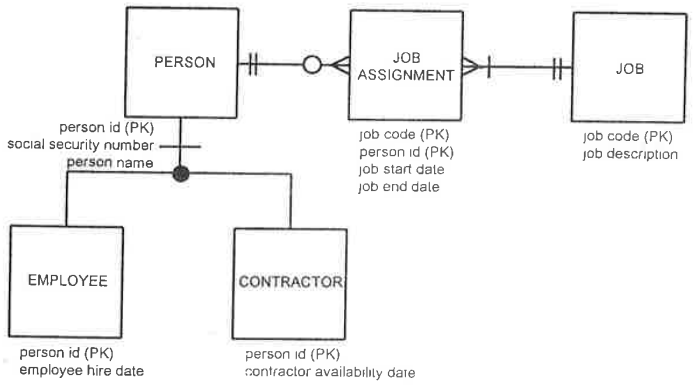


Figure 5.24 One possible resolution of cardinality conflict.

oped to identify and retain contractors who could remain on call to fulfill the assignment. Thus, every job had at least one person assigned to cover it. The personnel department also convinced the division staff that having every employee assigned to a job was not practical. This was especially true since one of their plants was closing down, and they had several employees going through their outplacement program.

Again, this is only one potential solution, and it is highly dependent on the definition of JOB, EMPLOYEE, and other components of the model. The key message here is that violating good practices of data modeling leads not only to information systems problems, but to missed opportunities to resolve underlying business conflicts.

Process Logic in the Data Model In this example, a fuel supply firm maintains supplies of different FUEL GRADEs at several small AIRPORTs in a large metropolitan area. The modeling team charged with helping the firm build a fuel management system has created the model depicted below. Based on input from the firm's managers, the team determined that several things occur with fuel at an airport—it is received, stored, and eventually dispensed. The model, they believe, accurately portrays this situation. They have added three relationships between AIRPORT and FUEL GRADE to portray the dynamics between airports and fuel. The key word here is "dynamics." A data model is a static view of the enterprise. The modeling team has done a very effective job of identifying the events that occur during the maintenance cycle at an airport. Thus, the model portrayed below includes information that should be recorded in the dynamic model of the fuel management system. From a data modeling perspective, it is important to understand that there exists a relationship between FUEL GRADE and AIRPORT. One characteristic of that relationship is that several events and processes occur involving both an airport and fuel. Further exploration of this model should lead you to question other aspects of its construction as well.

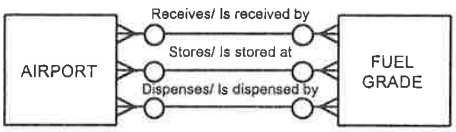


Figure 5.25 Multiple relationships representing process logic.

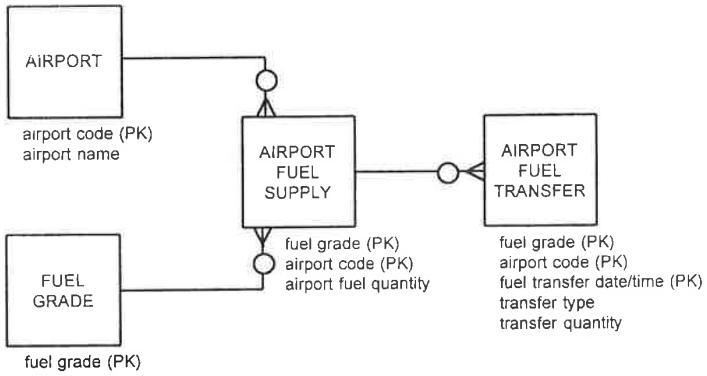


Figure 5.26 One possible resolution of multiple relationships.

In our hypothetical resolution, we have removed the process logic from the model, and returned it to its static state. This is, of course, only one possible solution.

Documenting Roles and Reasons Our final example examines the popular use of multiple relationships to document reasons or roles that entities play in relation to each other. Figure 5.26 documents a model fragment created by an airport management team. The team wanted to capture the business rules surrounding ownership, operation, and inspection of the airport by various interested business parties.

Look closely at the model in Figure 5.27. What does it really say about the relationship between AIRPORT and BUSINESS PARTY? It says that there must be at least one BUSINESS PARTY identified as an

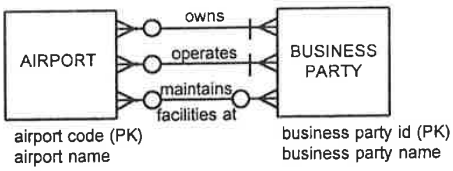


Figure 5.27 Multiple relationships for multiple reasons.

owner, and at least one identified as an operator for each AIRPORT. This set of relationships is further complicated by the fact that each is a many-to-many cardinality. In order to resolve this complex mix in an elegant fashion, the modeling team developed the solution shown in Figure 5.28. The team created AIRPORT BUSINESS PARTY to resolve the many-to-many cardinality between AIRPORT and BUSINESS PARTY (see Many-to-Many Relationships rule). In order to accommodate the multiple roles that a BUSINESS PARTY could have in relation to an AIRPORT, which were represented by the three relationships in our original example, the team created the entity AIRPORT BUSINESS PARTY ROLE, which defines a set of values pertaining to an AIRPORT BUSINESS PARTY, and whether that role is required or not required for an AIRPORT. The primary key of AIRPORT BUSINESS PARTY ROLE migrates to AIRPORT BUSINESS PARTY and is used as a primary key in that entity in combination with *airport code* and *business party id*. This key structure (in AIRPORT BUSINESS PARTY ROLE) allows an individual BUSI-

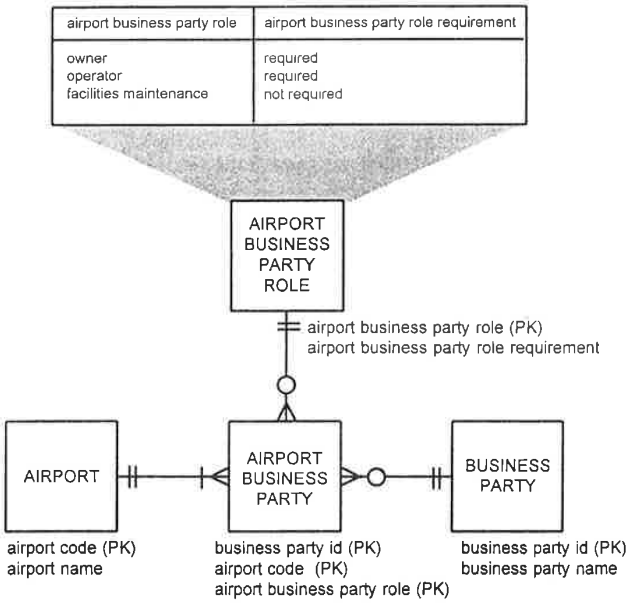


Figure 5.28 One possible resolution of multiple role problem.

NESS PARTY to perform more than one role in relation to an AIRPORT. The team included an attribute *airport business party role requirement* in AIRPORT BUSINESS PARTY ROLE in order to establish a business rule that an AIRPORT *must be* associated with a BUSINESS PARTY that assumes the role of "owner," and one that assumes the role of "operator." Therefore, when a new AIRPORT instance is created, we would expect at least one (although maybe more than one) instance of AIRPORT BUSINESS PARTY to be created with an *airport business party role* value equal to "owner" and one equal to "operator." The roles of "owner" and "operator" are required (as indicated in Figure 5.27 and the relationship cardinalities in Figure 5.28) for each AIRPORT.

5.9 PARALLEL ASSOCIATIVE ENTITIES

Parallel associative entities should not be permitted.

Description

As we discussed in the last rule, multiple relationships between two entities introduce some problems into the model. The same problems exist when "parallel" associative entities are introduced. Recall that an associative entity is created to resolve a many-to-many relationship between two entities. Parallel associative entities may creep into a model when more than one modeling team is working in the same subject area of the model, particularly when the teams have little or no intercommunication. The easiest way to identify this situation is from the identical key structures that exist. Parallel associative entities introduce the following problems into a data model:

- **Redundancy.** Each of the associative entities will carry the same set of inherited primary keys, and have the tendency to duplicate non-key information as well.
- Potential for **introducing contradictory business rules**, or for hiding conflicts in business rules that may exist in different parts of an enterprise. Remember that the data model is a key tool for identifying and resolving those conflicts to ensure consistent access to standard data products.
- Potential for **introducing consistency problems** into a database designed from the model.

The resolution of parallel associative entities is relatively straightforward—consolidate the associatives into a single associative entity. Follow the consolidation with a thorough examination of attribute edit

rules and ownership authorities, to ensure that adequate requirements information is recorded to manage data integrity and security.

Example

The example depicts a series of relationships existing for a firm that sells a host of products in its line of clothing outlets. The model shown below resulted from three separate data modeling efforts—a marketing team that decides which products will be sold at each outlet according to local demographics (OUTLET PRODUCT MIX), an inventory team that must plan product orders and order inventory accordingly (OUTLET PRODUCT INVENTORY), and a financial management team that needs to track the sales and cost performance of various products at different outlets (OUTLET PRODUCT COST). When the models were consolidated, the model management team recognized three parallel associative entities.

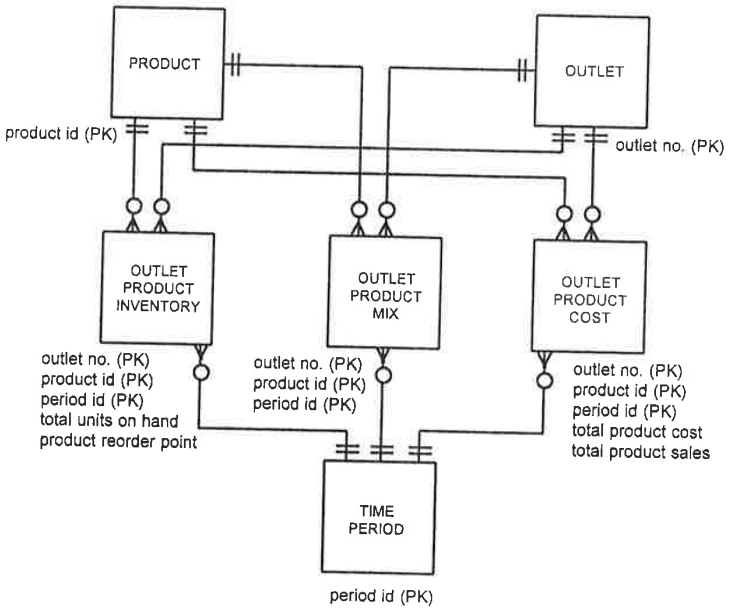


Figure 5.29 Unresolved parallel associative entities.

Note the potential for problems in this version of the model. First, the marketing team's decision to sell products at an outlet determines whether there will be any inventory or costs associated with the product at that outlet. Creation of instances of either OUTLET PRODUCT INVENTORY or OUTLET PRODUCT COST without a corresponding instance of OUTLET PRODUCT MIX is possible in the current model, but would be incorrect. Second, all three associative entities have the same key structure. This indicates, of course, that the non-key attributes of any of the three associative entities are actually functionally dependent on the primary key of all three associative entities. The model carries a clear normalization problem as a result. Finally, looking ahead to design and implementation, the creation of tables or other data structures to accommodate three entities with identical key structures in this case introduces unnecessary overhead into the system.

A recommended resolution to the multiple associative entity problem is depicted in Figure 5.30.

It is important to recognize that none of the departments has lost any information needed to perform its functions. The model is now correctly normalized (according to our assumptions), and can be implemented in one or more information systems.

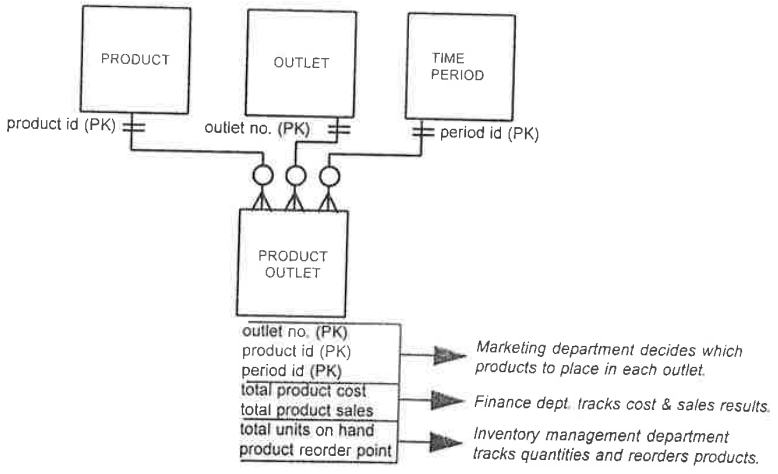


Figure 5.30 Resolved parallel associative entities.

5.10 RECURSIVE RELATIONSHIPS

Rule Resolve recursive relationships.

Discussion

A recursive relationship is one in which an occurrence of an entity is related to one or more other occurrences of the same entity. These are usually depicted with a relationship like that shown in Figure 5.31 in our example.

Example

Figure 5.31 depicts a recursive relationship from one instance of PERSON to another. A recursive relationship is often modeled this way initially.

Alternatively, we may first recognize recursive associations as relationships that exist among subtypes of the same supertype entity, as illustrated in Figure 5.32. The relationship between SUPERVISOR and EMPLOYEE implies a relationship between two occurrences of the PERSON entity, since each occurrence of EMPLOYEE and SUPERVISOR requires the existence of a corresponding occurrence of PERSON. From a business perspective, supervisors and employees have a personal relationship (or in some cases, an impersonal relationship!). Note in particular the key structure of EMPLOYEE. We have migrated the *person id* primary key from SUPERVISOR as a nonprimary foreign key in EMPLOYEE. That does not violate our rule that a subtype must have the same primary key as its parent, so we could leave the model as it currently stands, and probably not encounter any major problems.

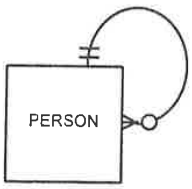


Figure 5.31 A typical recursive relationship notation.

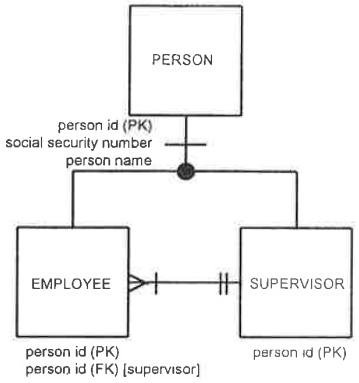


Figure 5.32 Relationships among subtypes of the same supertype.

But what if we have numerous subtypes of PERSON, each with relationships to the other? Our keys could be become increasingly difficult to handle. Or what if we have a matrix organization, where an employee could have several supervisors, or we wish to track an employee's supervisory history? Figure 5.33 shows our example modified to depict an associative entity between EMPLOYEE and SUPERVISOR.

This works if subtypes have been identified, and there are not too many of them. Recursive relationships come in all shapes and sizes, and the attempts to resolve them are often inconsistent even within the same model.

In order to deal with both simple and complex variations of recursive relationships quickly and consistently, we need a general scheme for resolving them. We have found Finkelstein's method to be very effective, and actually quite elegant (1989). Finkelstein establishes one of his business normal forms—fifth business normal form (5BNF) to be exact—to deal with recursive relationships. He defines 5BNF as:

> An entity is in fifth business normal form if its dependencies on occurrences of the same entity or entity type have been moved into a STRUCTURE entity.

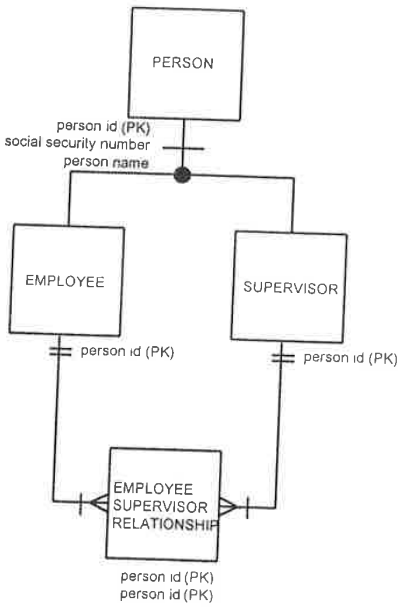
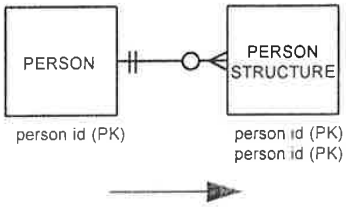


Figure 5.33 Associative entity between two subtypes.

A structure entity is a special type of attributive entity wherein two instances of the parent primary key are migrated to the child, rather than one. Additional keys are added if necessary to distinguish between multiple relationships between the same two occurrences. Figure 5.34 illustrates the general form of a structure entity primary key. Note that the presence of two instances of the *person id* primary key in PERSON STRUCTURE allows us to establish a relationship between two occurrences of PERSON. In fact, every occurrence in PERSON can be related to every other occurrence of PERSON through the PERSON STRUCTURE entity.

The structure entity is generally only used at the supertype level of a generalization-specialization hierarchy. There are several variations of the structure entity that can be used to handle special circumstances. We strongly recommend you read Finkelstein's discussions on structure entities (1989, 1992).



Two primary keys migrated to establish relationship between two instances of the same entity.

Figure 5.34 Migration of structure entity primary keys.

REFERENCES

Date, C. J. *An Introduction to Database Systems, Volume I.* Reading, MA: Addison-Wesley, 1986.

Finkelstein, Clive. *An Introduction to Information Engineering: From Strategic Planning to Information Systems.* Sydney: Addison-Wesley, 1989.

Finkelstein, Clive. *Information Engineering: Strategic Systems Development.* Sydney: Addison-Wesley, 1992.

FIPS PUB XXX, *Integration Definition for Information Modeling IDEF1X*, September 1992.

Korth, Henry F. and Silberschatz, Abraham. *Database System Concepts.* New York: McGraw-Hill, 1986.

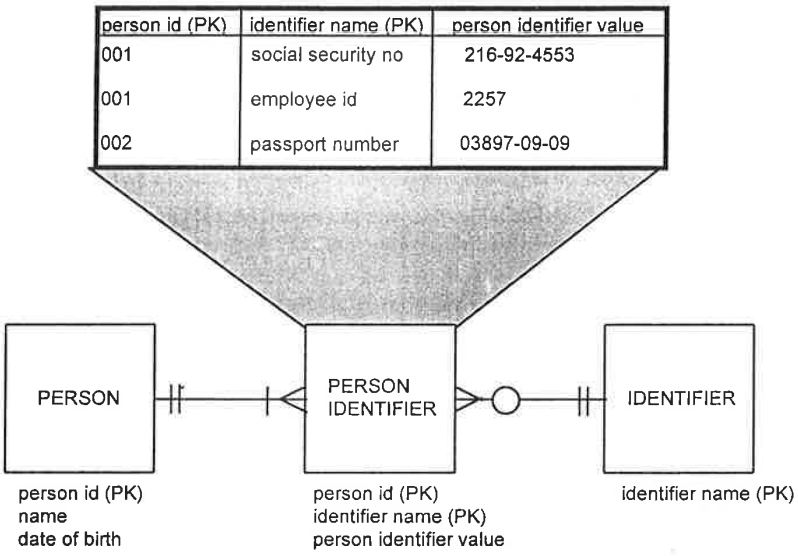


Figure 9.20 Multiple identifiers for a PERSON—Possible resolution.

additional occurrence without impact to the model. This model fragment may not communicate to business users as well as the previous model, but it certainly will be easier to maintain. All relationships that may be introduced to the subtypes in the previous model—a DRIVER drives a CAR, an EMPLOYEE works for an ORGANIZATION—will be maintained as optional relationships from PERSON. The person identifier value attribute maintains the value of each identifier as applies to an individual.

9.7 MODELING LOCATION

Discussion

There are almost as many ways to model location as there are data modelers. It is a concept that seems so simple on the surface, yet leads to many different personal opinions. The problem with LOCATION is that there are so many possible location objects, and then there are many permutations of these objects. For example, a location could contain x and y coordinates, room number, apartment number, PO Box,

street address, city, state, country, bed number, or a number of other possibilities. To complicate matters, the number of valid combinations of these objects is seemingly infinite. The following methods are presented to promote discussion.

Examples

The first method of modeling location is straightforward. A LOCATION is described with an artificial key, *location id*, and then a number of non-key attributes including *street address*, *city*, *state*, *zip code*, and *country*. PERSON LOCATION REASON describes the connection between PERSON and LOCATION. Is it a mailing address, billing address, home address, or work address? This allows us to distinguish among addresses. Figure 9.21 describes this method.

This is a rather simplistic attempt at modeling a LOCATION. It assumes all LOCATION occurrences have a *street address*, *city*, *state*, *zip code*, and *country*. Post office boxes, building numbers, room numbers are ignored in this model. Foreign locations and nonpostal schemes are ignored.

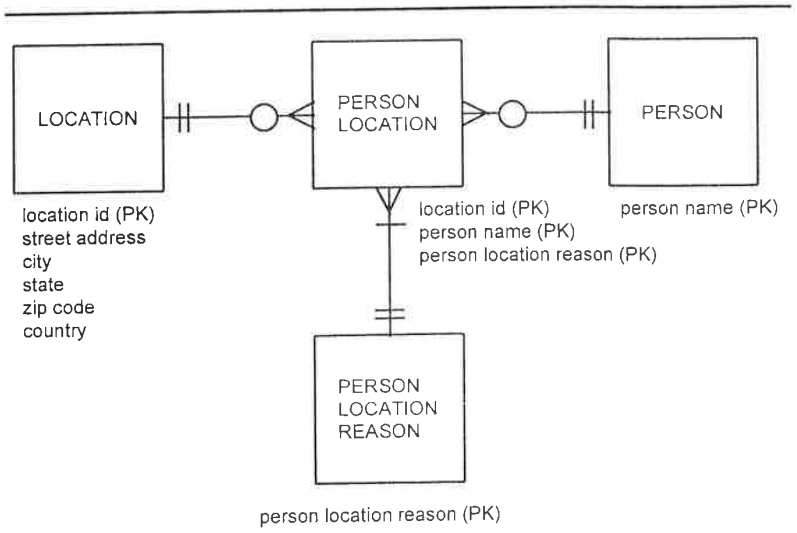


Figure 9.21 Location—Questionable method.

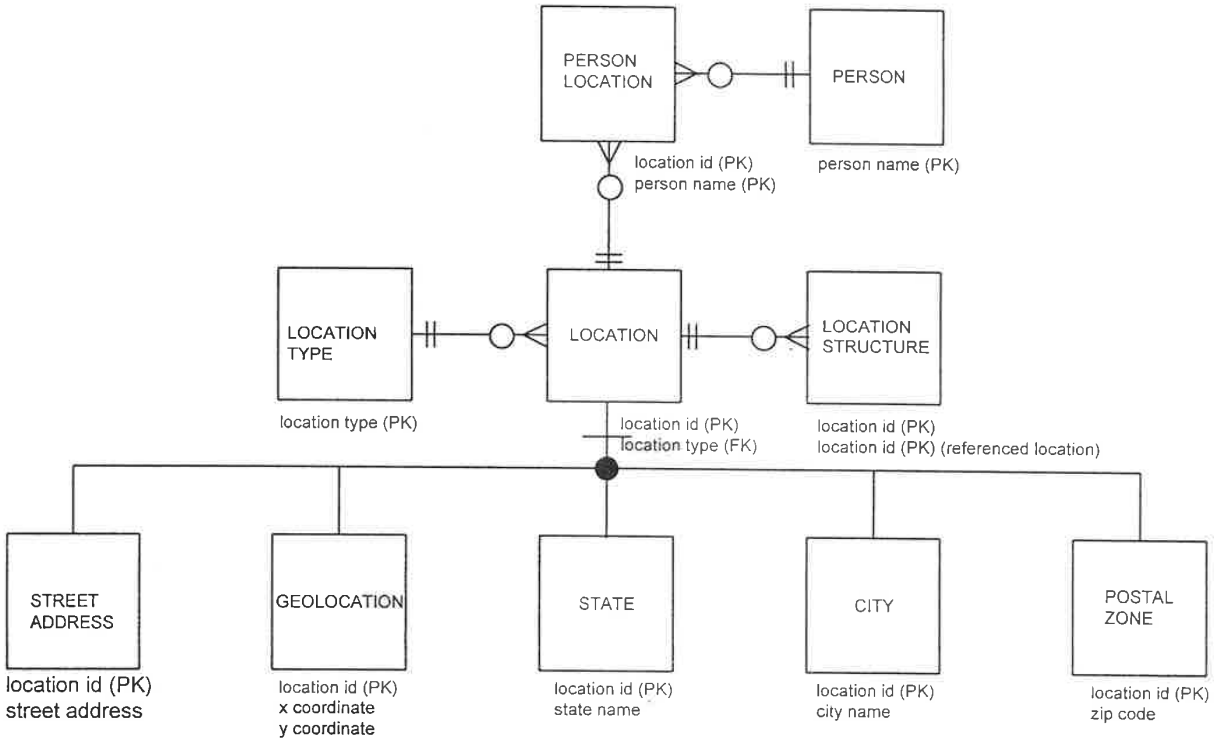


Figure 9.22 Location—One method.

As a second example, Figure 9.22 describes a method of modeling LOCATION that is straightforward and flexible. As a new location object is discovered, it is simply added as a subtype. In the example below, we have CITY, STATE, STREET ADDRESS, ZIP subtypes. LOCATION TYPE contains the valid values for the subtypes. LOCATION STRUCTURE shows the hierarchical relationship between locations—cities within states, street addresses within cities. If we wish to discover where a PERSON lives, we simply extract it from the PERSON LOCATION entity.

This is a structure that is easy to implement and flexible. Post office boxes, room numbers, building numbers and other mechanisms for identifying locations can simply be placed in the generalization structure. Nonpostal schemes for identifying locations are included as subtypes, such as GEOLOCATION.

In Figure 9.23, a more abstract method is utilized. A LOCATION is a coordinating mechanism. It contains no attributes other than an artificial key. It is associated with other entities in the data model to determine their location. A POSITIONING OBJECT identifies categories of locations such as a street, state, longitude, latitude, zip code, and so on. It is used to identify a COORDINATE. A COORDINATE is a symbol representing a place. It helps humans find where a specific place is. For example, it may be 2000 15th Street or Germany or 113 degrees longitude. Thus, a LOCATION can have many COORDINATES and a COORDINATE can have many LOCATIONS. NAVIGATION is a framework for interpreting a COORDINATE. Instances of NAVIGATION could be postal, grid, and the like. NAVIGATION identifies the different systems for identifying locations. Every POSITIONING OBJECT is supported by one or more methods of NAVIGATION. A NAVIGATION POSITIONING OBJECT entity allows many NAVIGATIONs to be associated with many OBJECTs.

9.8 MODELING BUSINESS (OR LEGAL) PARTY

Discussion

A business party is a generalization problem that is very often faced by the data modeler. It occurs when a business rule requires the participation of two seemingly unrelated entities. For example, a customer may be either a person or an organization. A legal party may be either a person or an organization. The enterprise maintains information about all of these entities and the data modeler is faced with a dilemma. How can they represent the similarities between the two entity types, while still maintaining their independence?